

A note on the security of the h HB protocol

Carl Löndahl

carl@grocid.net

July 28, 2015

Abstract

We propose a polynomial-time attack on the h HB protocol, showing that the protocol does not attain the security it claims. Our attack is based on the attack introduced in [2].

1 Introduction

In the modern era of cryptography, researchers have struggled with finding low-cost cryptographic primitives suitable for simplistic hardware environments such as RFID tags and low-power/cost devices. A popular source of inspiration is the *Learning Parity with Noise* (LPN) problem, which has roots in machine learning theory but now has gained a lot of popularity among cryptographers. The LPN problem is strongly related to the problem of *decoding random linear codes*, which probably is the most important problem in coding theory. Being supposedly hard, LPN plays an important role in post-quantum cryptography in contrast to classic number theoretic problems. LPN consists of very basic arithmetic operations and is therefore a perfect fit for light-weight cryptography.

The first ‘real’ cryptographic construction based on LPN was the Hopper-Blum (HB) protocol [5] – a minimalistic protocol being secure in a *passive* attack model. Juels and Weis [6], and Katz and Shin [7] proposed a modified protocol, HB^+ , which aimed to be secure also in the *active* attack model by extending HB with one extra round. However, Gilbert *et al.* [2] later showed that the HB^+ protocol is vulnerable to active attacks, i.e., man-in-the-middle attacks. Later, Gilbert *et al.* [3] proposed a variant of the Hopper-Blum protocol called $HB^\#$.

Some of the more recent contributions to LPN-based constructions are a two-round identification protocol called Lapin, proposed by Heyse *et al.* [4], and an LPN-based encryption scheme called HELEN, proposed by Duc and Vaudenay [1]. The Lapin protocol is based on an LPN variant called Ring-LPN, where the samples are elements of a polynomial ring.

Khoureich proposed in [8] a new version of HB called h HB (*harder HB*), which aims to repair susceptibility against GRS attacks. In this paper, we show that this is not the case.

2 The h HB protocol

The author of [8] argues that the weakness of HB is due to that the secret x does not change over time. The h HB protocol is built upon the hypothesis; by employing a way of (presumably) securely transmitting a secret session value and letting x take this value, [8] aims to patch this weakness. Once the session value has been transmitted, the verifier in the h HB protocol runs the standard HB protocol to verify the tag. The h HB protocol is outlined in Protocol 1.

Protocol 1 (h HB protocol)

Tag(s, y)		Reader(s, y)
<hr/>		
		$\tau \xleftarrow{\$} \{0, 1\}, \xi_0 \xleftarrow{\$} \{0, 1\}, \xi_1 \xleftarrow{\$} \{0, 1\}$
$(\tau, \xi_0, \xi_1) \leftarrow f_s^{-1}(\alpha, \beta, \gamma, 0^{ s })$	$\xleftarrow{(\alpha, \beta, \gamma)}$	$(\alpha, \beta, \gamma) \leftarrow f_s(\tau, \xi_0, \xi_1, 0^{ s })$
$\theta \leftarrow \xi_\tau$		$\theta \leftarrow \xi_\tau$
$p_0 \leftarrow \theta^{ s }$		$p_0 \leftarrow \theta^{ s }$
<hr/>		
(Repeat k times)		$\tau \xleftarrow{\$} \{0, 1\}, \xi_0 \xleftarrow{\$} \{0, 1\}, \xi_1 \xleftarrow{\$} \{0, 1\}$
$(\tau, \xi_0, \xi_1) \leftarrow f_s^{-1}(\alpha, \beta, \gamma, p_{i-1})$	$\xleftarrow{(\alpha, \beta, \gamma)}$	$(\alpha, \beta, \gamma) \leftarrow f_s(\tau, \xi_0, \xi_1, p_{i-1})$
$x_i \leftarrow \xi_\tau$		$x_i \leftarrow \xi_\tau$
$p_{i-1} \leftarrow x_1 x_2 \dots (x_i)^{ s -i+1}$		$p_{i-1} \leftarrow x_1 x_2 \dots (x_i)^{ s -i+1}$
<hr/>		
(Repeat r times)		
$x \leftarrow x_1 x_2 \dots x_k$		$x \leftarrow x_1 x_2 \dots x_k$
$b \xleftarrow{\$} \{0, 1\}^k$	\xrightarrow{b}	
	\xleftarrow{a}	$a \xleftarrow{\$} \{0, 1\}^k$
$\nu \leftarrow \text{Ber}_\epsilon$		
$z \leftarrow a \cdot x \oplus b \cdot y \oplus \nu$	\xrightarrow{z}	Verify $a \cdot x \oplus b \cdot y = z$
<hr/>		

The function used by the reader to transmit session values τ, ξ_0, ξ_1 ,

$$f_s(\lambda_1, \lambda_2, \lambda_3, p_i) \rightarrow (\alpha, \beta, \gamma) \quad (1)$$

is defined in Algorithm 1. Similarly, the inverse function used by the tag to decode session values τ, ξ_0, ξ_1 ,

$$f_s^{-1}(\alpha, \beta, \gamma) \rightarrow (\lambda_1, \lambda_2, \lambda_3, p_i) \quad (2)$$

is given in Algorithm 2.

3 The attack

We will now proceed with describing our attack. We use a method very similar to that of [2].

Algorithm 1 (function f_s)

Input: $\lambda_1, \lambda_2, \lambda_3 \in \{0, 1\}$, $p_i \in \{0, 1\}^k$	1 $c_1 \xleftarrow{\$} \{0, 1\}^k, t_1 \leftarrow c_1 \cdot (s \oplus p_i) \oplus \lambda_1$ 2 $c_2 \xleftarrow{\$} \{0, 1\}^k, t_2 \leftarrow c_2 \cdot (s \oplus p_i) \oplus \lambda_2$ 3 $c_3 \xleftarrow{\$} \{0, 1\}^k, t_3 \leftarrow c_3 \cdot (s \oplus p_i) \oplus \lambda_3$ Output: Triple (α, β, γ)
	4 if $\lambda_1 \oplus \lambda_2 \oplus \lambda_3 = 0$ then 5 return $((c_3, t_3), (c_1, t_1), (c_2, t_2))$ 6 else 7 return $((c_2, t_2), (c_3, t_3), (c_1, t_1))$

Algorithm 2 (function f_s^{-1})

Input: Triple (α, β, γ)	1 $(c_1, t_1) \leftarrow \alpha, (c_2, t_2) \leftarrow \beta, (c_3, t_3) \leftarrow \gamma$ 2 $\lambda_1 \leftarrow c_1 \cdot (s \oplus p_i) \oplus t_1$ 3 $\lambda_2 \leftarrow c_2 \cdot (s \oplus p_i) \oplus t_2$ 4 $\lambda_3 \leftarrow c_3 \cdot (s \oplus p_i) \oplus t_3$ Output: $\lambda_1, \lambda_2, \lambda_3 \in \{0, 1\}$,
	5 if $\lambda_1 \oplus \lambda_2 \oplus \lambda_3 = 0$ then 6 return $(\lambda_2, \lambda_3, \lambda_1)$ 7 else 8 return $(\lambda_3, \lambda_1, \lambda_2)$

3.1 Determining secret y

The tag sends the following

$$a \cdot x \oplus b \cdot y \oplus \nu = z$$

The verifier checks if the following is satisfied

$$a \cdot x \oplus b \cdot y \stackrel{?}{=} z,$$

which is expected to be true for $r \cdot \mathbb{E}(\nu = 0) = r \cdot \epsilon$ of the r samples. If the number of correct equations are above some threshold, the verifier **accepts**. Otherwise the verifier **rejects**.

First, we ignore the x vector. By intercepting the communication between the tag and the verifier, we are able to perturb the interchanged bits. To determine the value of bit of y at index i , we run the following steps.

1. Let the tag and verifier exchange the session value. For now, this is ignored.
2. When the tag sends b , we flip the i th bit. So,

$$b'_i \leftarrow b_i \oplus 1.$$

3. Then, we let the tag and verifier run the r steps. If the reader returns **accept**, then the bit y_i is very likely to be 0. Naturally, we may amplify the probability of a correct guess by re-running the procedure for the same bit b_i .
4. By repeating for all k bits, we can determine the secret value y .

3.2 Determining secret s

The second stage of the attack aims to determine the secret vector s . In the very first step of exchanging the session value p_0 is always $0^{|s|}$, which is the key to our exploit. To determine the value of bit of s at index j , we run the following steps.

1. In the first step of the session-value exchange, flip the j th bit in c_1 . As a result, we have

$$\lambda_1 \leftarrow (c_1 \oplus \delta_j) \cdot s \oplus t_1. \quad (3)$$

where δ_j is a vector with 1 on index j and all-zero on the remaining indices. Hence,

$$\lambda_1 \leftarrow \begin{cases} \lambda_1 & \text{if } s_j = 0, \\ \lambda_1 \oplus 1 & \text{if } s_j = 1. \end{cases}$$

Applying the same procedure to c_2 and c_3 , we are able to conditionally flip also λ_2 and λ_3 .

2. If the two values satisfy $\xi_0 = \xi_1$ (which is true with probability $\frac{1}{2}$), then x will be perturbed at position 0, i.e., $x_0 \leftarrow x_0 \oplus 1$. Everything else remains the same.
3. When the tag is verified against the reader, we set the j th bit of a to always be 1. Hence, the tag will compute

$$z \leftarrow \begin{cases} a \cdot x \oplus b \cdot y \oplus \nu & \text{if } s_j = 0, \\ a \cdot x \oplus b \cdot y \oplus \nu \oplus \phi & \text{if } s_j = 1. \end{cases} \quad (4)$$

where $\mathbb{P}(\phi = 1) = \frac{1}{2}$. So, if $s_j = 1$, then the reader will output **reject** with probability $\frac{1}{2}$. Running the procedure a polynomial number of times for the same index j will give a good estimate of s_j .

Implementation of hHB and the MITM attack can be found at [9].

References

- [1] A. Duc and S. Vaudenay. HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In *In proceedings of AFRICACRYPT 2013*, pages 107–126, 2013.

- [2] H. Gilbert, M. J. B. Robshaw, and Y. Seurin. An active attack against hb^+ —a provably secure lightweight authentication protocol. In *Cryptology ePrint Archive, Report 2005/237*, 2005. <http://eprint.iacr.org/>.
- [3] H. Gilbert, M. J. B. Robshaw, and Y. Seurin. $HB^\#$: Increasing the Security and the Efficiency of HB^+ . In N. P. Smart, editor, *Advances in Cryptology—EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer-Verlag, 2008.
- [4] S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. An Efficient Authentication Protocol Based on Ring-LPN. In *Fast Software Encryption 2012*, volume 4965 of *Lecture Notes in Computer Science*, pages 346–365. Springer-Verlag, 2012.
- [5] N. J. Hopper and M. Blum. Secure human identification protocols. In C. Boyd, editor, *Advances in Cryptology—ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2001.
- [6] A. Juels and S. A. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology—CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer-Verlag, 2005.
- [7] J. Katz and J. S. Shin. Parallel and concurrent security of the hb and hb^+ protocols. In S. Vaudenay, editor, *Advances in Cryptology—EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 73–87. Springer-Verlag, 2006.
- [8] K. A. Khourreich. hHB : a Harder HB^+ Protocol. *12th International Conference on Security and Cryptography (SECRYPT 2015)*, to appear, July 2015.
- [9] C. Löndahl. *Github*. <https://github.com/grocid/hhb>.